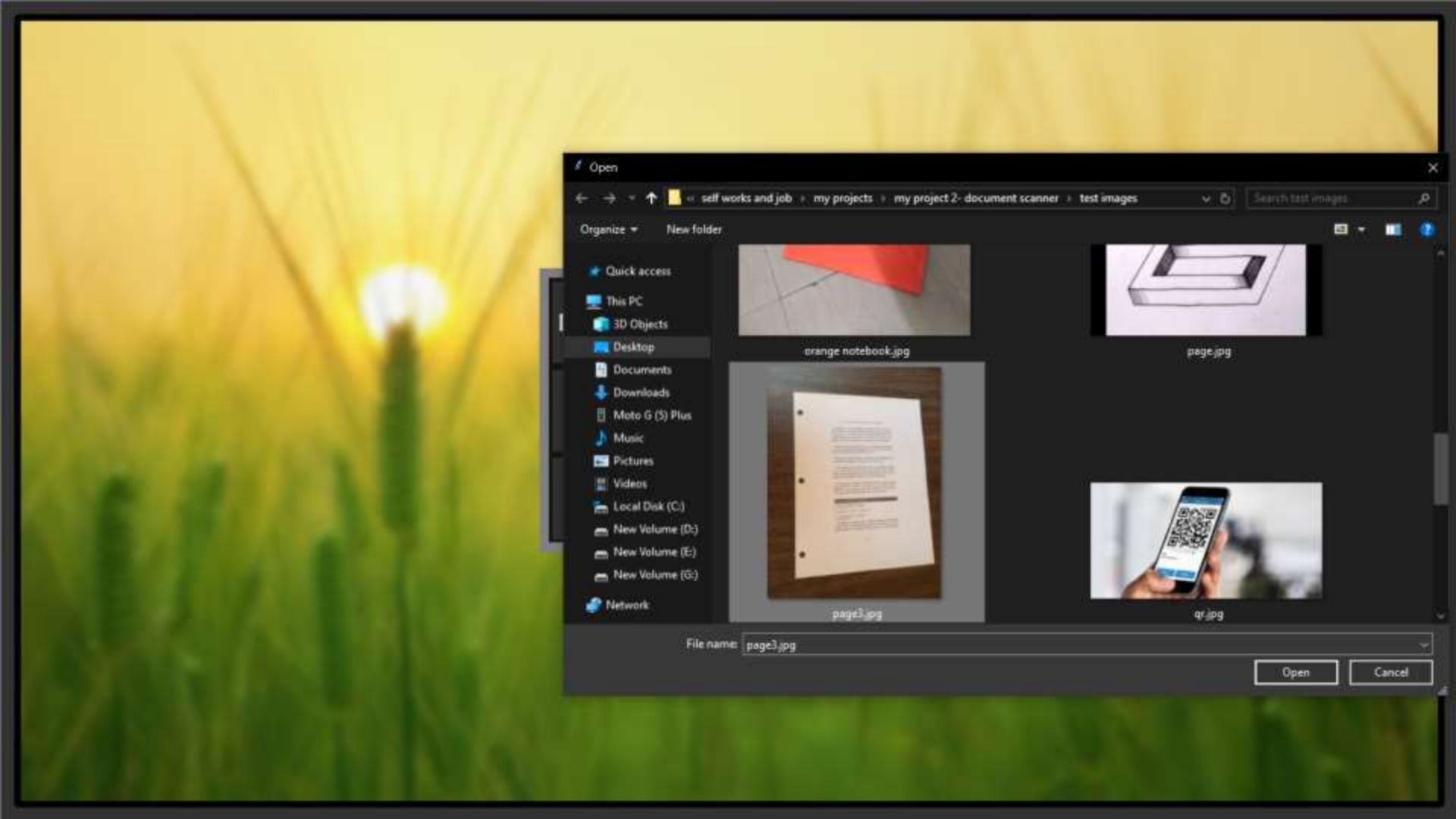
Document Scanner Application


Document Perspective Corrector
Image QR code detector
Exit


Open
self works and job
my projects
my project 2- document scanner
test images
Search test images
Organize
New folder
Quick access
This PC
3D Objects
Desktop
Documents
Downloads
Moto G (5) Plus
Music
Pictures
Videos
Local Disk (C:)
New Volume (D:)
New Volume (E:)
New Volume (G:)
Network
orange notebook.jpg
page.jpg
page3.jpg
qr.jpg
File name: page3.jpg
Open
Cancel

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0,0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, not a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

Listing 4.3: getting_and_setting.py

```
17 corner = image[0:100, 0:100]
18 cv2.imshow("Corner", corner)
19
20 image[0:100, 0:100] = (0, 255, 0)
21
22 cv2.imshow("Updated", image)
23 cv2.waitKey(0)
```

On **line 17** we grab a 100 × 100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0, 0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, not a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

Listing 4.3: getting_and_setting.py

```
(corner = image[0:100, 0:100]
cv2.imshow("Corner", corner)

image[0:100, 0:100] = (0, 255, 0)

cv2.imshow("Updated", image)
cv2.waitKey(0)
```

On **line 17** we grab a 100 × 100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

X: 1012 Y: 406

Main Menu

Next

Rotate

Done

Reset

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0, 0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, not a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

Listing 4.3: getting_and_setting.py

```
17 corner = image[0:100, 0:100]
18 cv2.imshow("Corner", corner)
19
20 image[0:100, 0:100] = (0, 255, 0)
21
22 cv2.imshow("Updated", image)
23 cv2.waitKey(0)
```

On **line 17** we grab a 100 × 100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

X: 1133 Y: 687

Main Menu

Next

Rotate

Done

Reset

X: 1133 Y: 687

Main Menu

Next

Rotate

Done

Reset

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0, 0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, not a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

Listing 4.3: getting_and_setting.py

```
(corner = image[0:100, 0:100]
cv2.imshow("Corner", corner)

image[0:100, 0:100] = (0, 255, 0)

cv2.imshow("Updated", image)
cv2.waitKey(0)
```

On **line 17** we grab a 100 × 100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0, 0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, not a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

Listing 4.2: getting_and_setting.py

```
11 corner = image[0:100, 0:100]
12 cv2.imshow("Corner", corner)
13
14 image[0:100, 0:100] = (0, 255, 0)
15
16 cv2.imshow("Updated", image)
17 cv2.waitKey(0)
```

On **line 17** we grab a 100 × 100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0, 0) and set it to have a value of `(0, 0, 255)`. If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, not a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

Listing 4.3: getting_and_setting.py

```
11 corner = image[0:100, 0:100]
12 cv2.imshow("Corner", corner)
13
14 image[0:100, 0:100] = (0, 255, 0)
15
16 cv2.imshow("Updated", image)
17 cv2.waitKey(0)
```

On **line 17** we grab a 100×100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0,0) and set it to have a value of `(0, 0, 255)`. If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, *not* a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

Listing 4.3: getting_and_setting.py

```
11 corner = image[0:100, 0:100]
12 cv2.imshow("Corner", corner)
13
14 image[0:100, 0:100] = (0, 255, 0)
15
16 cv2.imshow("Updated", image)
17 cv2.waitKey(0)
```

On **line 11** we grab a 100×100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0, 0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, *not* a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

Listing 4.3: getting_and_setting.py

```
11 corner = image[0:100, 0:100]
12 cv2.imshow("Corner", corner)
13
14 image[0:100, 0:100] = (0, 255, 0)
15
16 cv2.imshow("Updated", image)
17 cv2.waitKey(0)
```

On **line 11** we grab a 100×100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0, 0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, not a blue color.

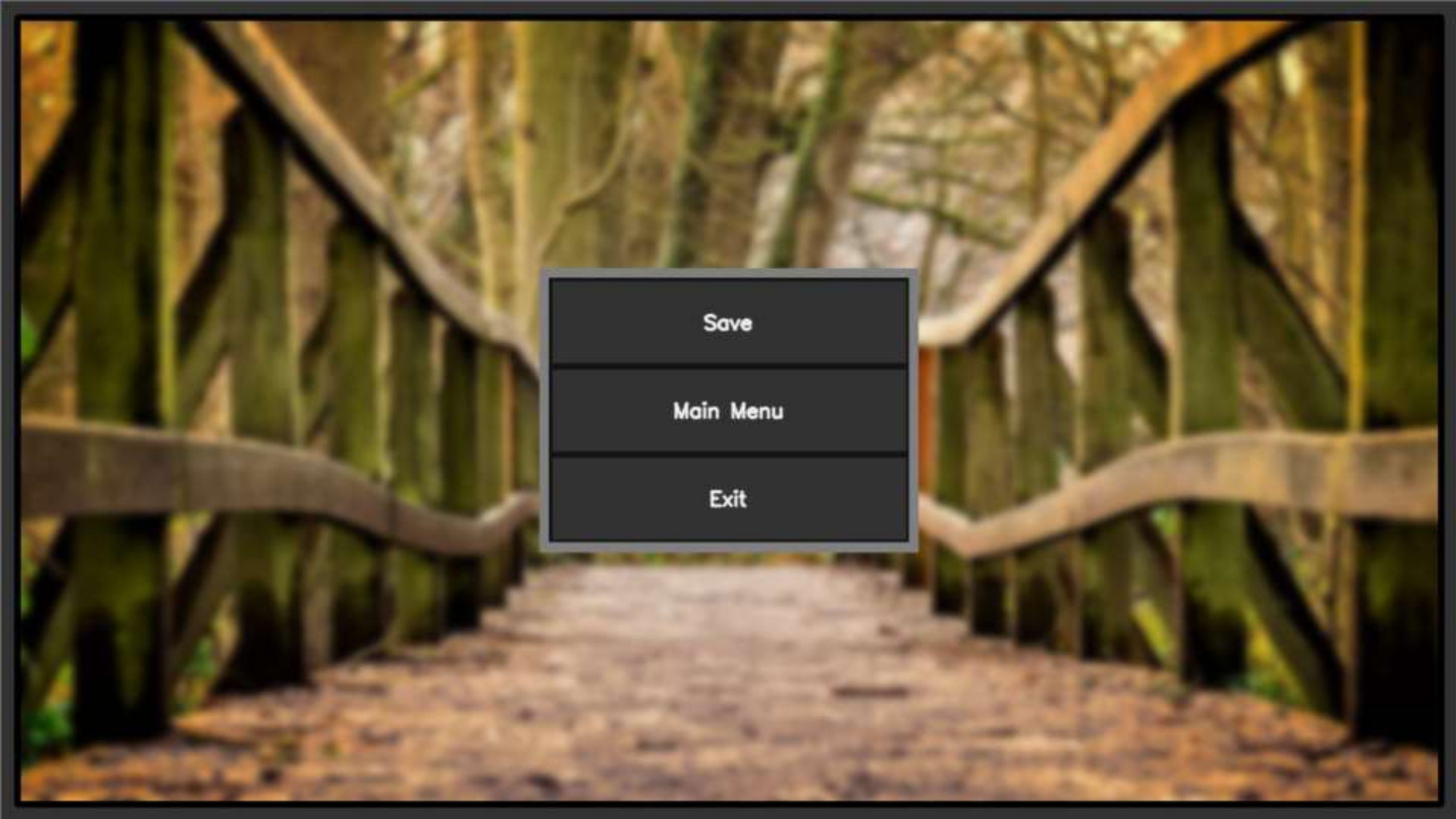
After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

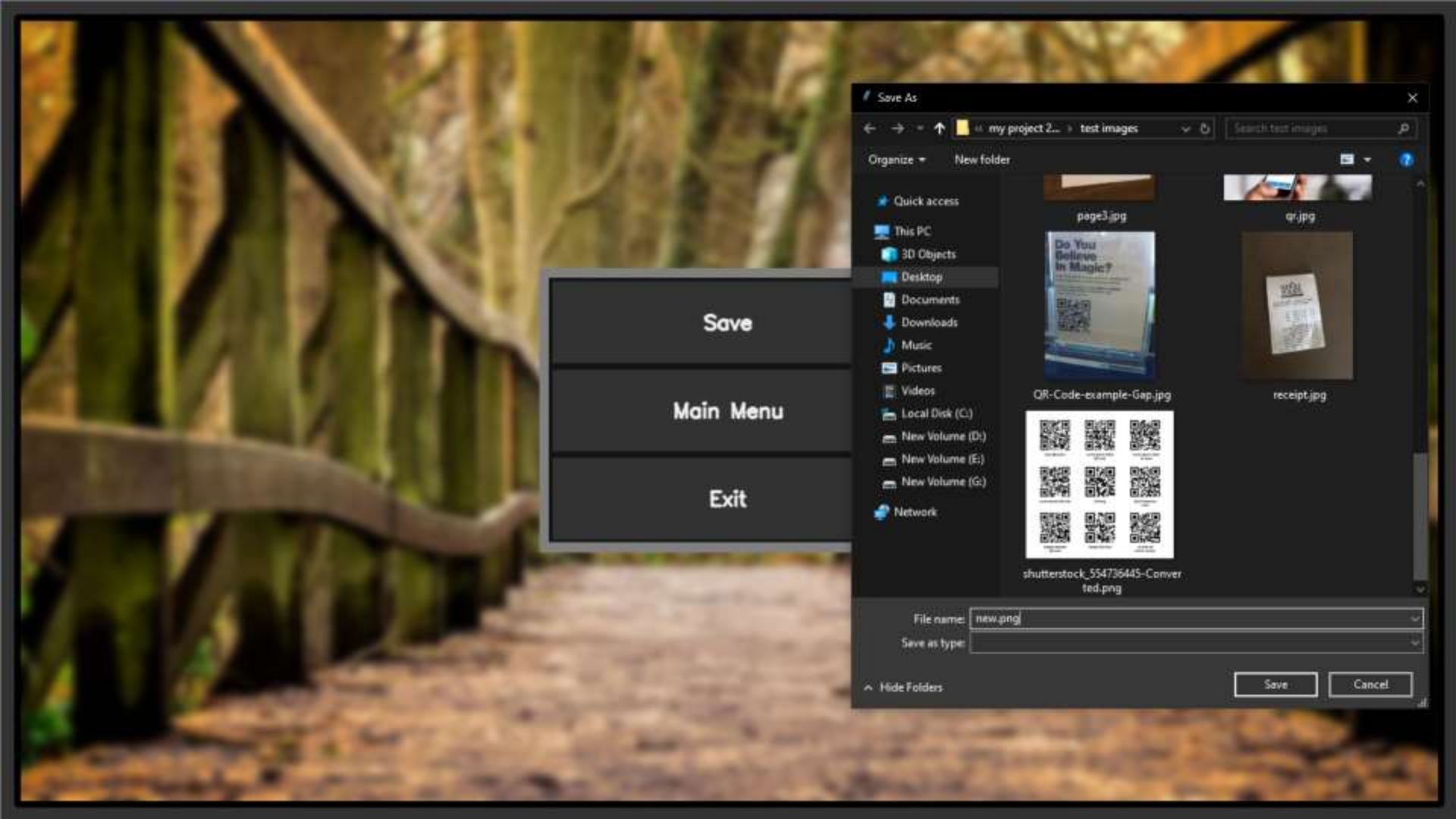
Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

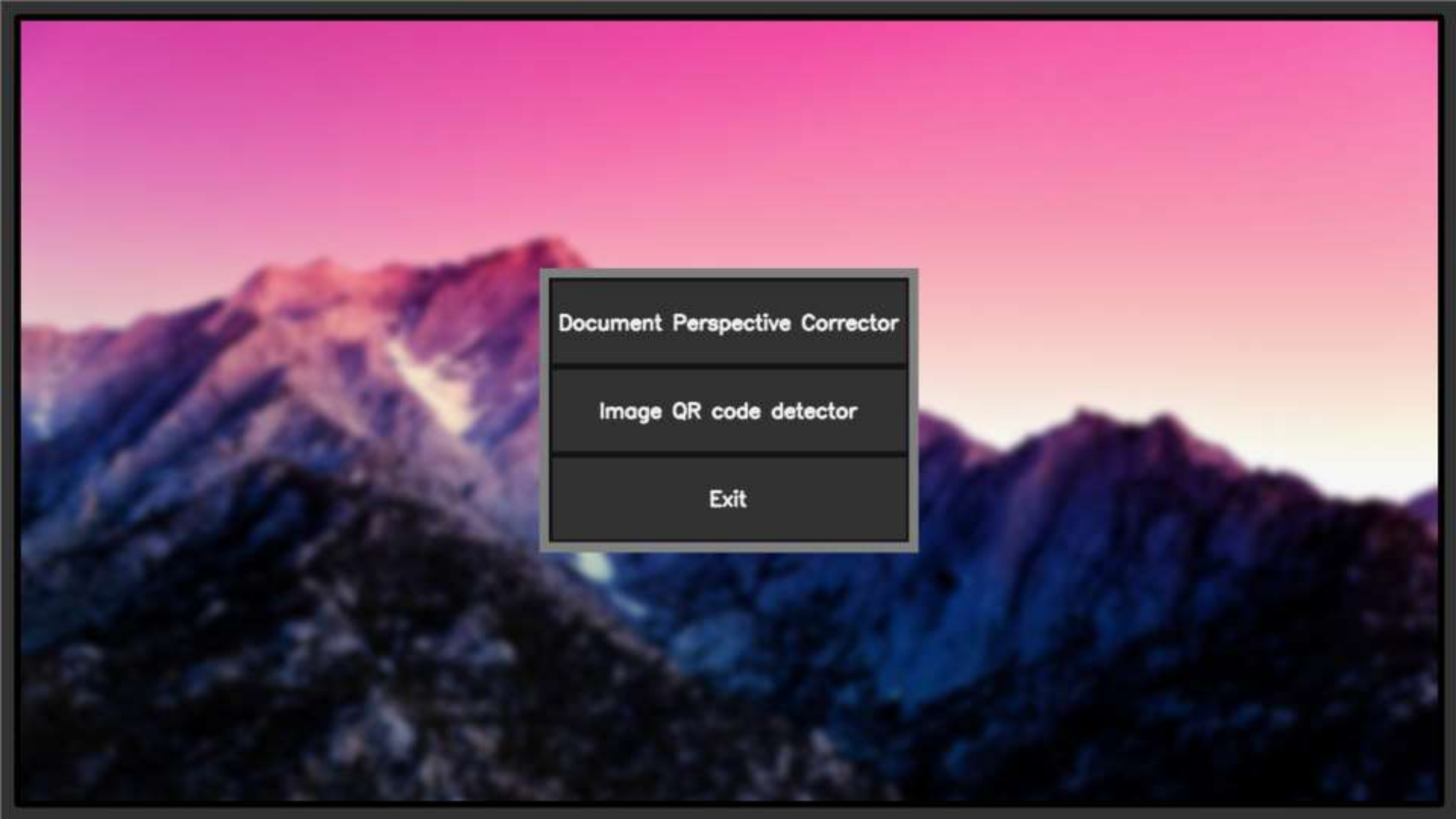
Listing 4.3: getting_and_setting.py

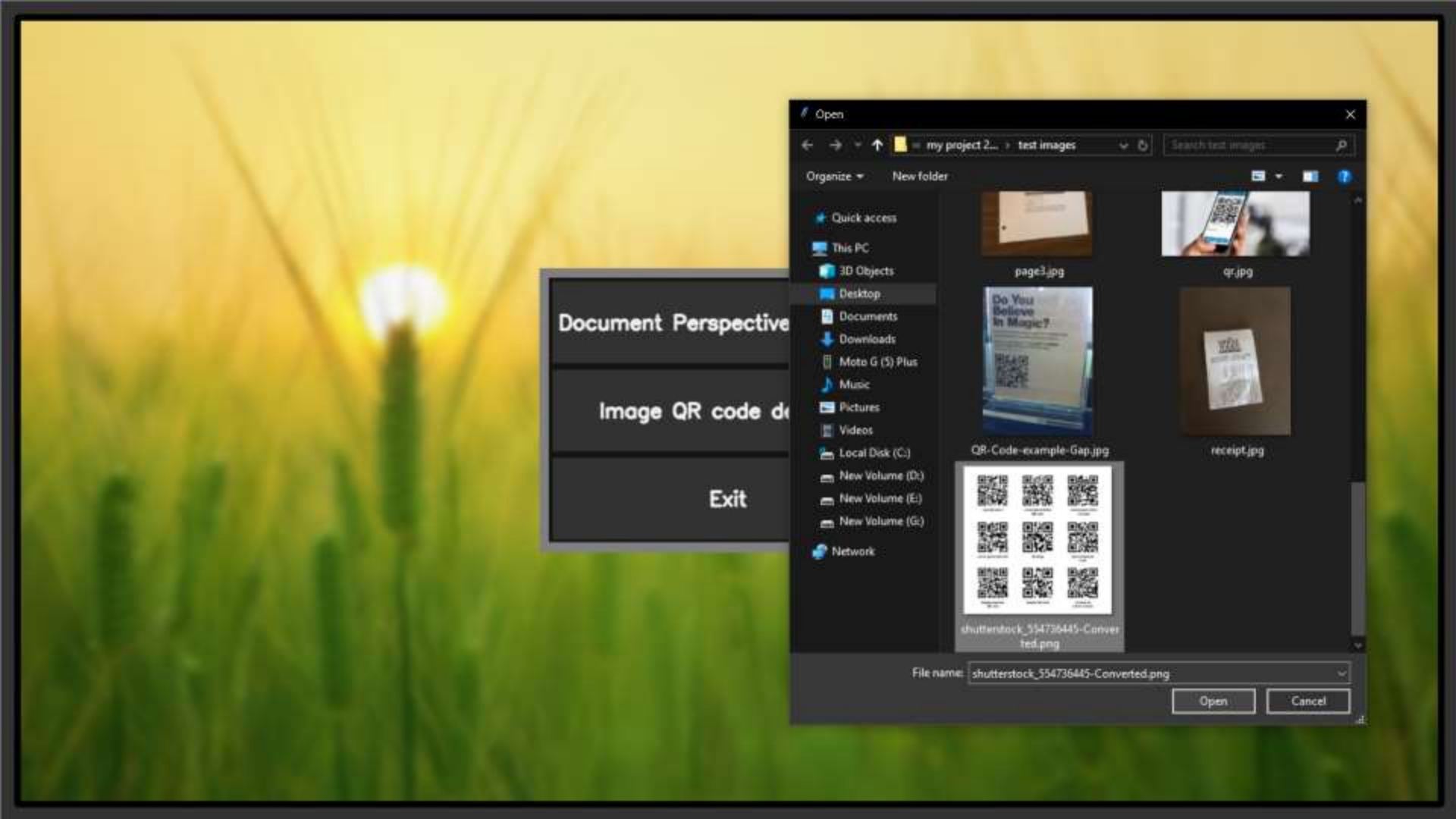
```
13 corner = image[0:100, 0:100]
14 cv2.imshow("Corner", corner)
15
16 image[0:100, 0:100] = (0, 255, 0)
17
18 cv2.imshow("Updated", image)
19 cv2.waitKey(0)
```

On **line 17** we grab a 100 × 100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four


Save
Main Menu
Exit


Save
Main Menu
Exit
Save As
my project 2...
test images
Search test images
Organize
New folder
Quick access
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
New Volume (D:)
New Volume (E:)
New Volume (G:)
Network
page3.jpg
qr.jpg
Do You Believe In Magic?
QR-Code-example-Gap.jpg
receipt.jpg
shutterstock_554736445-Converted.png
File name:
new.png
Save as type:
Hide Folders
Save
Cancel


Document Perspective Corrector
Image QR code detector
Exit


Document Perspective
Image QR code de
Exit
Open
my project 2... › test images
Search test images
Organize
New folder
Quick access
This PC
3D Objects
Desktop
Documents
Downloads
Moto G (5) Plus
Music
Pictures
Videos
Local Disk (C:)
New Volume (D:)
New Volume (E:)
New Volume (G:)
Network
page3.jpg
qr.jpg
QR-Code-example-Gap.jpg
receipt.jpg
shutterstock_554736445-Converted.png
File name: shutterstock_554736445-Converted.png
Open
Cancel

Main Menu | Copy Data | Please click one of the red box to see the data here and press Copy Data button to copy it to clipboard.

! Just QR-code !

Lorem Ipsum Dolor QR-code

Lorem ipsum dolor sit amet

Lorem Ipsum QR-code

Nothing

Quick Response Code

Sample abstract QR-code

Sample QR-code

Ut enim ad minim veniam

Main Menu | Copy Data | Lorem Ipsum Dolor QR-code

! Just QR-code !

Lorem Ipsum Dolor QR-code

Lorem ipsum dolor sit amet

Lorem Ipsum QR-code

Nothing

Quick Response Code

Sample abstract QR-code

Sample QR-code

Ut enim ad minim veniam

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0, 0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, *not* a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

Listing 4.3: getting_and_setting.py

```
17 corner = image[0:100, 0:100]
18 cv2.imshow("Corner", corner)
19
20 image[0:100, 0:100] = (0, 255, 0)
21
22 cv2.imshow("Updated", image)
23 cv2.waitKey(0)
```

On **line 17** we grab a 100 × 100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four